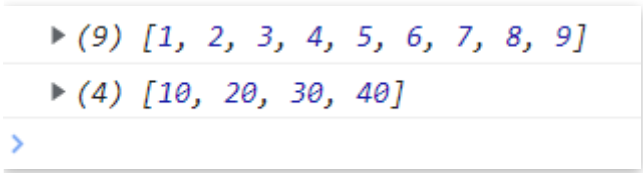
1、对象拓展

```
<script>
        //1. Object.is 判断两个值是否完全相等
        console.log(Object.is(120, 120)); // true
        console.log(Object.is(NaN, NaN)); // true
        console.log(NaN === NaN); // false

        //2. Object.assign 对象的合并
        const config1 = {
            host: 'localhost',
            port: 3306,
            name: 'root',
            pass: 'root',
            test: 'test'
        };
        const config2 = {
            host: 'http://atguigu.com',
            port: 33060,
            name: 'atguigu.com',
            pass: 'iloveyou',
            test2: 'test2'
        }
        console.log(Object.assign(config1, config2));

        //3. Object.setPrototypeOf 设置原型对象  Object.getPrototypeof
        const school = {
            name: 'guang'
        }
        const cities = {
            xiaoqu: ['北京', '上海', '深圳']
        }
        Object.setPrototypeOf(school, cities);
        console.log(Object.getPrototypeOf(school));
        console.log(school);
    </script>
```

2、数组拓展

```
<script>
        //flat 扁平化
        //将多维数组转化为低位数组
        const arr = [1, 2, 3, 4, [5, 6]];
        const arr1 = [1, 2, 3, 4, [5, 6, [7, 8, 9]]];
        //参数为深度 是一个数字
        console.log(arr1.flat(2));

        //flatMap
        const arr2 = [1, 2, 3, 4];
        const result = arr2.flatMap(item => [item * 10]);
        console.log(result);
    </script>
```



3、试题

1、['1', '2', '3'].map(parseInt) 输出什么?

```
 /**parseInt（string,radix）解析一个字符串并返回指定基数的十进制整数，radix是2-36之间的整数，表示被解析的字符串的基数
        string :
```

```
            要被解析的值。如果参数不是一个字符串，则将其转换为字符串。字符串开头的空白符将会被忽略
        radix（可选）：
            从2到36，表示字符串的基数。类如，指定16表示被解析值是十六进制数。请注意，10不是默认值
        返回值：
            从给定的字符串中解析出一个**整数**，但是当radix小于2或者大于36，或者第一个非空格字符不能转换为数字，返回NaN
    */
    ([1,2,3].map(parseInt));//[1,NaN,NaN]
    //相当于：
    [1,2,3].map((item,i)=>{})
        parseInt('1',0)       //1
        parseInt('2',1)       //NaN
        parseInt('3',2)       //NaN  二进制（0,1）所以 3=> NaN
```

2、New操作符具体干了什么?

```
1．创建了一个全新的对象。
2．将对象链接到这个函数的prototype对象上。
3．执行构造函数，并将this绑定到新创建的对象上。
4．判断构造函数执行返回的结果是否是引用数据类型，若是则返回构造函数执行的结果，否则返回创建的对象。
```

3、什么是浅拷贝，深拷贝以及和他们之间的区别

```
浅拷贝：是创建一个新对象，这个对象有着原始对象属性值的一份精确拷贝。
如果属性是基本类型，拷贝的就是基本类型的值，如果属性是引用类型，拷贝的就是内存地址。
深拷贝：是将一个对象从内存中完整的拷贝一份出来,从堆内存中开辟一个新的区域存放新对象。
区别：浅拷贝基本类型之前互不影响，引用类型其中一个对象改变了地址，就会影响另一个对象;
      深拷贝改变新对象不会影响原对象，他们之前互不影响。

浅拷贝和深拷贝的区别：
浅拷贝：
  浅拷贝是会将对象的每个属性进行依次复制，但是当对象的属性值是引用类型时，实质复制的是其引用，当引用指向的值改变时也会跟着变化。
深拷贝：
  深拷贝复制变量值，对于非基本类型的变量，则递归至基本类型变量后，再复制。 深拷贝后的对象与原来的对象是完全隔离的，互不影响， 对一个对象的修改并不会影响另一个对象
```

浅拷贝实现方法：

```
// 1 赋值
        let person = {
            name: 'aaa',
            age: 19
        }

        let obj = person;
        obj.name = 'bbb';
        obj.age = 20;
        console.log(obj, person);
```

深度复制：

```
递归实现深拷贝思路：
将要拷贝的数据 obj 以参数的形式传参
声明一个变量 来储存我们拷贝出来的内容
判断 obj 是否是引用类型数据，如果不是，则直接赋值即可（ 可以利用 obj instanceof Type 来进行判断），
由于用 instanceof 判断array 是否是object的时候，返回值为true，所以我们在判断的时候，直接判断obj 是否是Array 就可避免这个问题
根据判断的不同类型，再给之前的变量赋予不同的类型： [ ] : { }
循环obj 中的每一项，如果里面还有复杂数据类型，则直接利用递归再次调用copy函数
最后 将 这个变量 return 出来即可

    <script>
        // 1 遍历对象
        let person = {
            name: 'aaa',
            age: 19
        }

        function copy(object) {
            let obj = {}
            for (let key in object) {
                obj[key] = object[key]
            }
            return obj;
        }
        let obj1 = copy(person)
```

```
        obj1.name = "bbb";
        obj1.age = 20;
        console.log(obj1, person);

        // 2 解构赋值
        let obj2 = {
            a: 1,
            b: 2,
            c: 3
        }

        let obj3 = { ...obj2 };
        obj3.b = 20;
        console.log(obj3, obj2);

        // 2 Object.assign()
        var obj4 = {
            name: "张三",
            age: 20,
            speak: function () {
                console.log("我是" + this.name);
            }
        };

        var obj5 = Object.assign({}, obj4);

        // 当修改obj5的属性和方法的时候，obj4相应的属性和方法不会改变
        obj5.name = "李四";
        console.log(obj4, obj5);

        // 3 递归实现深拷贝
        var obj4 = {   //原数据，包含字符串、对象、函数、数组等不同的类型
            name: "test",
            main: {
                a: 1,
                b: 2
            },
            fn: function () {

            },
```

```
            friends: [1, 2, 3, [22, 33]]
        }

        function copy(obj4) {
            let newobj = null;    //声明一个变量用来储存拷贝之后的内容

            // 判断数据类型是否是复杂类型，如果是，则调用自己，再次循环，如果不是，直接赋值即可，
            // 由于null不可以循环但类型又是object，所以这个需要对null进行判断
            if (typeof (obj4) == 'object' && obj4 !== null) {

                // 声明一个变量用以储存拷贝出来的值,根据参数的具体数据类型声明不同的类型来储存
                newobj = obj4 instanceof Array ? [] : {};

                // 循环obj4 中的每一项，如果里面还有复杂数据类型，则直接利用递归再次调用copy函数
                for (var i in obj4) {
                    newobj[i] = copy(obj4[i])
                }
            } else {
                newobj = obj4
            }
            return newobj;     //函数必须有返回值，否则结构为undefined
        }

        var obj5 = copy(obj4)
        obj5.name = '修改成功'
        obj5.main.a = 100
        console.log(obj4, obj5)

        // 4 JSON 实现深度复制
        let a = { name: 'andy', age: 20, address: 'AAA' };
        let b = JSON.parse(JSON.stringify(a));
        b.age = 21;
        console.log(b);
    </script>
```

4、console.log (!!(new Boolean(false)) 输出什么?

```
true // new Boolean(false) new了一个布尔实例对象，！！后为true
```

5、在数组的原型上实现indexOf方法

```
Array.prototype.myIndexOf = function (searchItem, startIndex) {
            let result = -1;
            let iStart = 0;
            if (startIndex >= 0 && startIndex < this.length) {//传了第二个值
                iStart = startIndex;
            } else if (startIndex >= this.length) {
                return -1
            }
            for (let i = iStart; i < this.length; i++) {
                if (this[i] === searchItem) {
                    result = i;
                    break;
                }
            }

            return result;
        }

        let str = ['a', 'b', 'c'];
        console.log(str.myIndexOf('a'));
        let arr1 = [1, 1, 2, 3];
        console.log(arr1.myIndexOf(1));
        console.log(arr1.myIndexOf(3, -2));
        console.log(arr1.myIndexOf(4));
```

```
0                                                    35_试题1.html:83
0                                                    35_试题1.html:85
3                                                    35_试题1.html:86
-1                                                   35_试题1.html:87
```

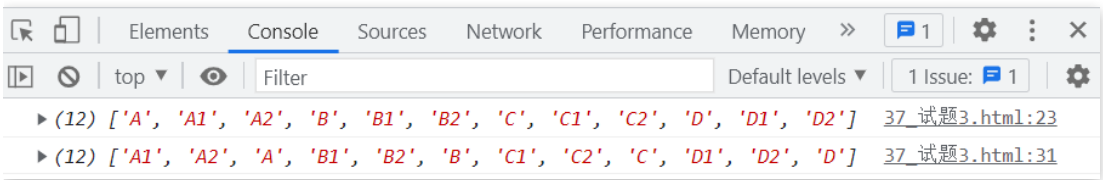
6、打印1000之内的所有对称数

```
function getNum(num) {
    let arr = [];
    for (let i = 10; i <= num; i++) {
        if (i = i.toString().split('').reverse().join('')) {
            console.log(i);
```

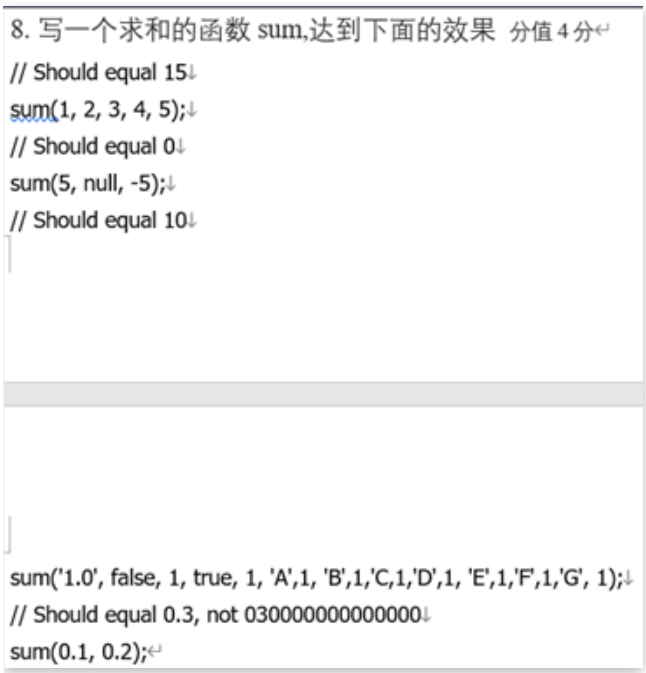
```
            arr.push(i);
        }
    }
    return arr;
}
```

7、 请把数组['A1', 'A2', 'B1', 'B2', 'C1', 'C2', 'D1', 'D2'] 和['A', 'B', 'C', 'D']，合并为['A1', 'A2', 'A', 'B1', 'B2', 'B', 'C1', 'C2', 'C', 'D1', 'D2', 'D']

```
        let arr1 = ['A1', 'A2', 'B1', 'B2', 'C1', 'C2', 'D1', 'D2'];
        let arr2 = ['A', 'B', 'C', 'D'];
        let arr3 = [...arr1, ...arr2].sort();
        console.log(arr3);

        arr3.sort((a, b) => {
            if (a.charAt(0) == b.charAt(0) && a.length > b.length) {
                return -1;
            };
        });

        console.log(arr3);
```



8、写一个求和的函数sum,达到下面的效果



```
function sum(...n) {
            let result = 0;
            // console.log([...arguments]);
            // console.log(n);
            n.forEach(item => {
                if (item !== true && item * 1) {
                    result += item;
                }
            })
            return result;
        }

        console.log(sum(12, 3, 4, 5,));
        console.log(sum(1, true, 2, 3));
        console.log(false * 1);//0
        console.log(true * 1);//1
        console.log(sum(5, null, -5));
```

```
24                                  36_试题2.html:41
6                                   36_试题2.html:42
0                                   36_试题2.html:43
1                                   36_试题2.html:44
0                                   36_试题2.html:45
```

9、new操作符具体干了什么呢?

49.new操作符具体干了什么呢?

1、创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。

2、属性和方法被加入到 this 引用的对象中。

3、新创建的对象由 this 所引用，并且最后隐式的返回 this 。

10、window onload事件和img或者其他媒体资源的onload加载事件的区别

img.onload 图片节点加载完毕不会调用 要资源加载完毕就会调用

window.onload：等待页面所有资源下载完成才执行,包括图片资源的下载，所以它是最慢的

网页加载顺序：url-->下载页面-->domTree，cssTree并行-->渲染树renderTree-->绘制页面-->继续下载图片资源，下载完毕再放到页面上去 onload

domTree：domTree的形成，是先把元素翻译成的节点对象挂到 domTree上去，再把属性 img_src放到渲染树上去

每年都考的面试题：用户从地址栏输入网址按下回车到页面展示出来 整个过程发生了什么?

答案:前端=>网络=>后端=>网络=>前端 这4步都得分析